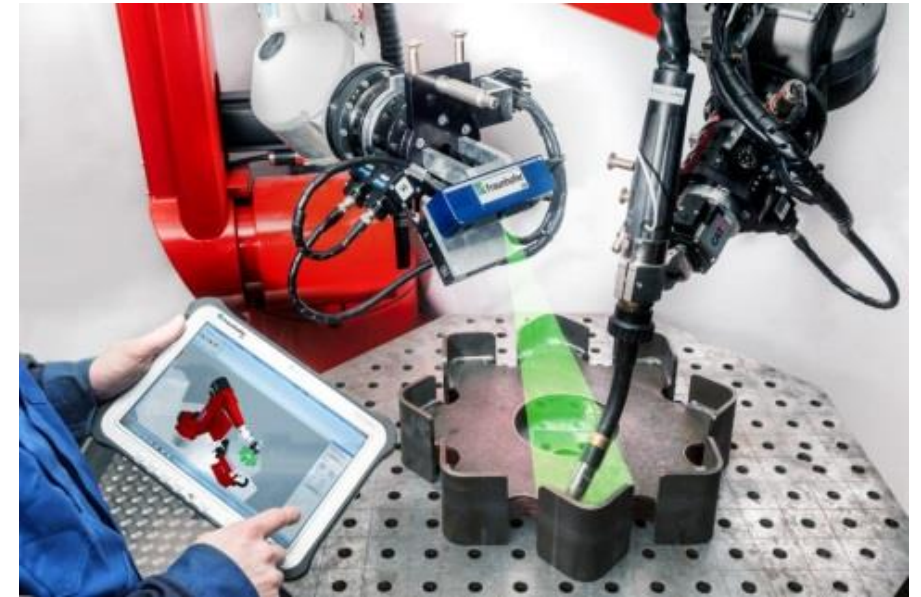
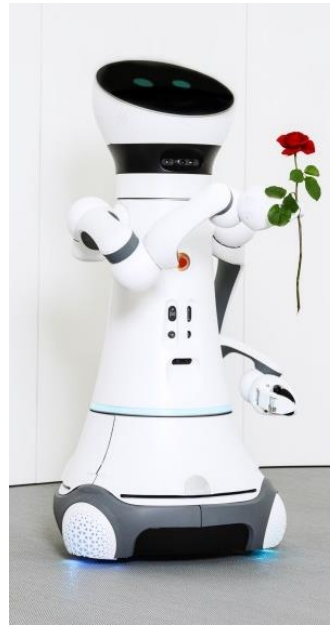


ROS-MODEL

COMBINE MDE WITH ROS EVERYDAY CODE

Nadia Hammoudeh Garcia



WHY? ROS lacks -> MDE advantages



- ✓ Fast prototyping
- ✓ Maximally flexible
- ✓ Collaborative environment
- ✓ Hardware independent
- ✓

- ✗ No quality standards
- ✗ No specifications enforced
- ✗ Lots of manual code -> Quality depends on the developer
- ✗ No validation at design time -> Runtime test

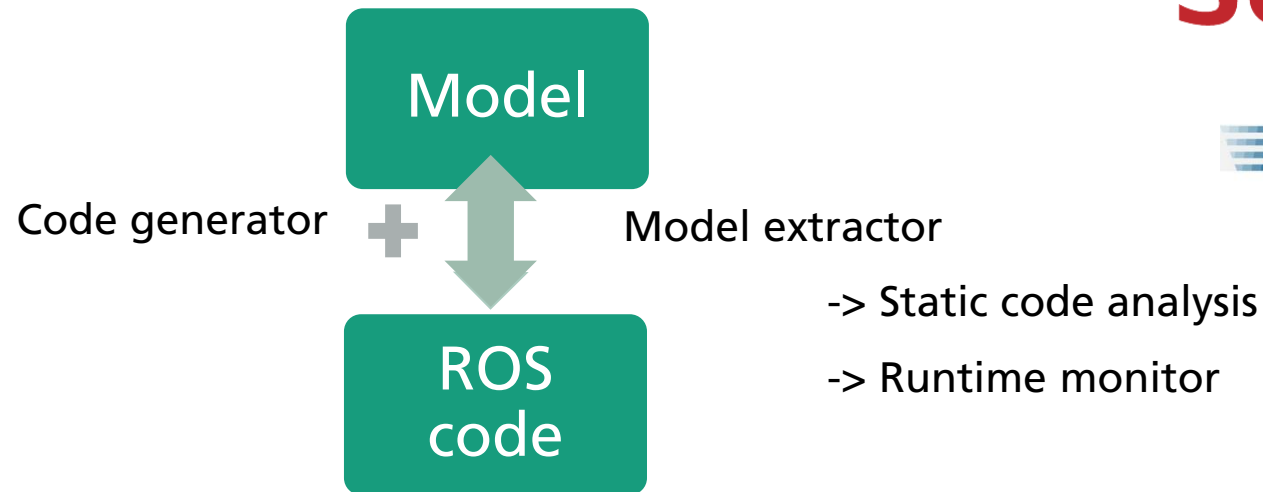


Model-Driven-Engineering potential benefits?

- Design patterns in application domains
- Model checker techniques
- Generation of code -> Less error-prone
-

Combine MDE and ROS? New?

- Traditional MDE approach Model-to-text to ensure code quality but for ROS case...
 - Boilerplate code low acceptance by ROS community
 - Hard to create and maintain THE code template
 - This approach ignores the 4000 hand-written open-source ROS packages



Gefördert durch:



aufgrund eines Beschlusses
des Deutschen Bundestages

Automatic extraction of ROS models – for single nodes and full systems

- Static code analysis: <https://github.com/git-afsantos/haros>

- Extract information without executing the code
- Support single nodes and launch files



- Runtime monitor: https://github.com/ipa-led/ros_graph_parser

- Extract model from a running systems
- Parser for rosgaph



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 723658.

Contribution

1) Models (Ecore format)

ROS Metamodel

- ROS package information
- Nodes and their interfaces (topics, services, actions)
- Communication objects (msgs, srvs, actions types)

ROSSystem Metamodel

- Groups of nodes
- Declare namespaces
- Remap interfaces

(roslaunch file information)



Contribution

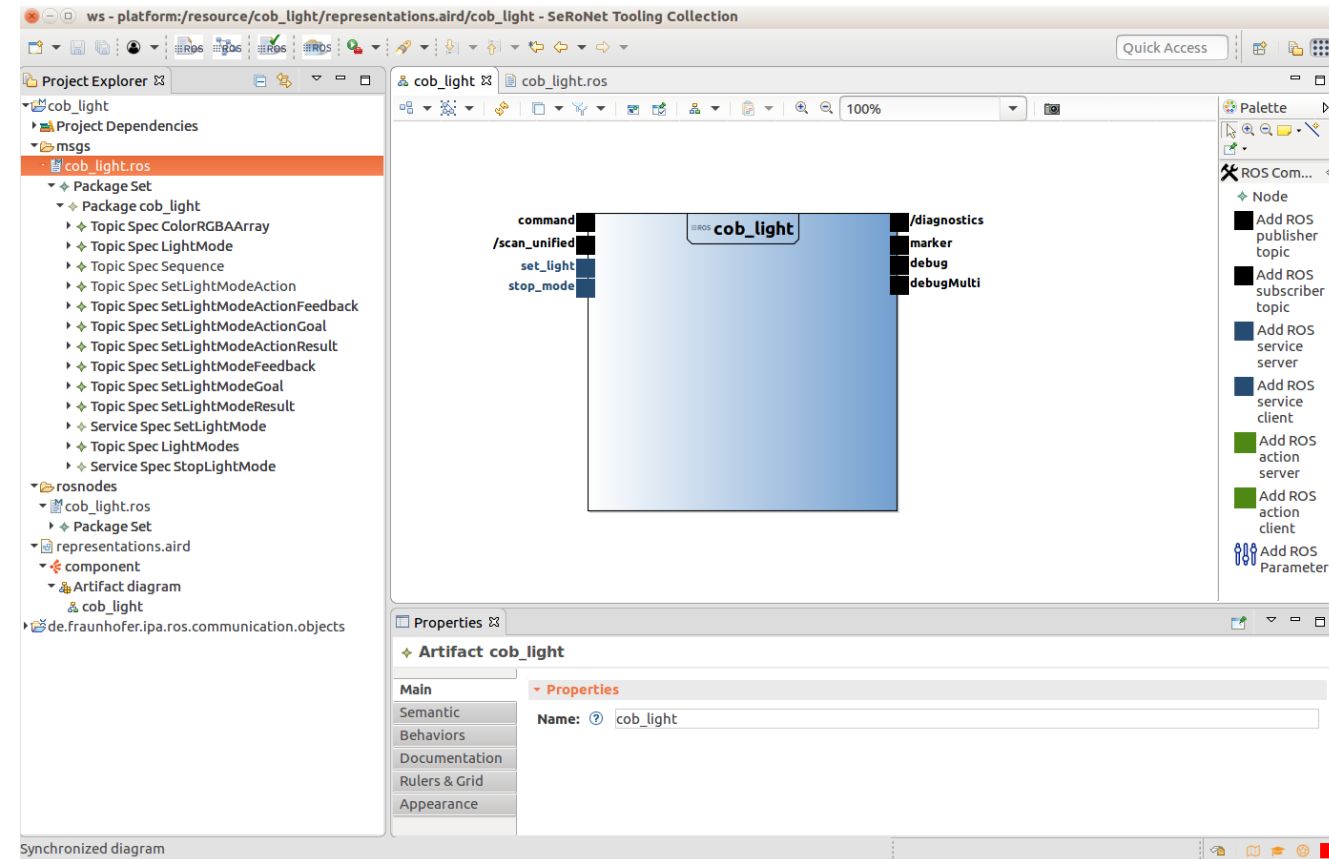
2) Tooling (eclipse based)



- Graphical interface
- Graphical model editors
- DSLs associated to the models
 - Languages parser, typechecker and editor
 - Rules for models validation
 - Compiler to generate code

- Release available

-> <http://ros-model.seronet-project.de/updatesite/>



Contribution

3) HAROS extractor as web service

Web interface: <http://ros-model.seronet-project.de>

- ROS node extractor:
 - Generates the equivalent RC
- Launch file extractor:
 - Generates RosSystem model

For private packages: Docker config
[cloud](#)

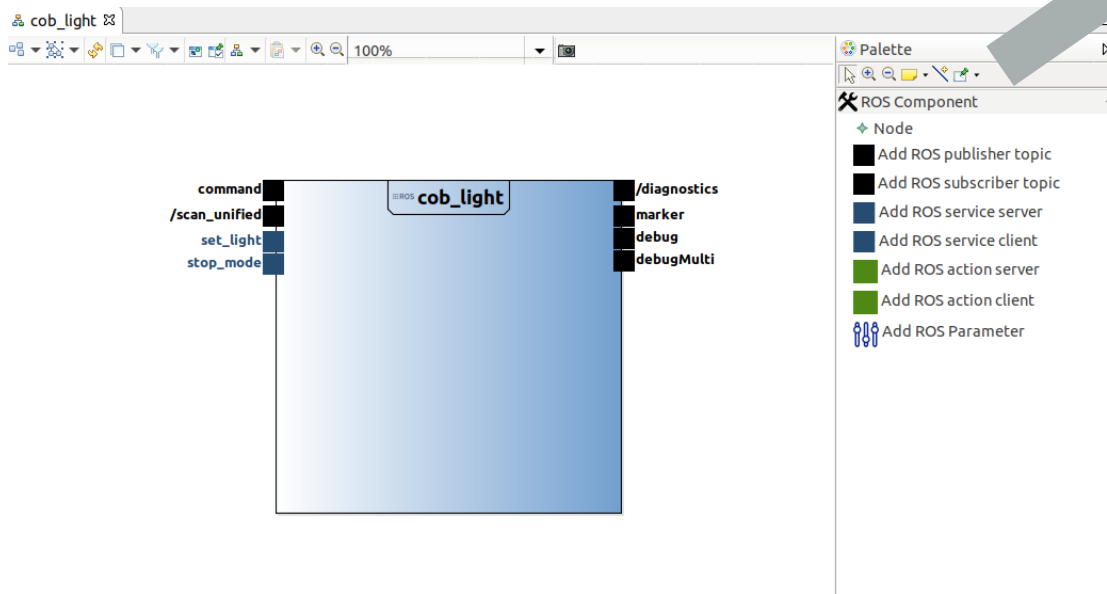
The screenshot shows the 'ROS Model Extractor' web interface. At the top, there are logos for 'SeRONet' and 'HAROS'. The interface is divided into two main sections: 'NODE ANALY...' and 'LAUNCH ANA...'. The 'NODE ANALY...' section contains input fields for 'Git repository' (https://github.com/ipa320/cob_driver), 'Package name' (cob_sick_s300), and 'Node name' (cob_sick_s300). A 'SUBMIT' button is at the bottom left. The 'LAUNCH ANA...' section is currently active, showing a 'DOWNLOAD THE FILES' button and the resulting ROS file 'cob_sick_s300.ros'. The content of this file is displayed as a PackageSet with a CatkinPackage and an Artifact, containing a Node with a publisher for 'scan' and 'scan_standby' messages, and a diagnostic publisher.

```
PackageSet { package {
  CatkinPackage cob_sick_s300 { artifact {
    Artifact cob_sick_s300 {
      node Node { name cob_sick_s300
        publisher {
          Publisher { name 'scan' message 'sensor_msgs.LaserScan'},
          Publisher { name 'scan_standby' message 'std_msgs.Bool'},
          Publisher { name '/diagnostics' message 'diagnostic_msgs.DiagnosticArray'}}
        }
      }
    }
  }
}
```

Benefits and applications

1) "ROS1" and ROS2 code generator

- "Boilerplate" code generator(templates for C++)
- Extract model from "ROS1" code and auto generate its equivalent model for ROS2



```
cob_light cob_light.cpp
}

class cob_light : public rclcpp::Node {
public:
  cob_light() : Node("cob_light") {
    diagnostics_ = this->create_publisher<diagnostic_msgs::msg::DiagnosticArray>("/diagnostics",10);
    marker_ = this->create_publisher<visualization_msgs::msg::Marker>("marker",10);
    debug_ = this->create_publisher<std_msgs::msg::ColorRGBA>("debug",10);
    debugMulti_ = this->create_publisher<cob_light::msg::ColorRGBAArray>("debugMulti",10);
    command_ = this->create_subscription<cob_light::msg::ColorRGBAArray>("command", 10, std::bind(&cob_light::command_callback, this, _1));
    scan_unified_ = this->create_subscription<sensor_msgs::msg::LaserScan>("/scan_unified", 10, std::bind(&cob_light::scan_unified_callback, this, _1));
    set_light_ = this->create_service<cob_light::srv::SetLightMode>("set_light", std::bind(&cob_light::set_light_handle, this, _1, _2));
    stop_mode_ = this->create_service<cob_light::srv::StopLightMode>("stop_mode", std::bind(&cob_light::stop_mode_handle, this, _1, _2));

    timer_ = this->create_wall_timer(500ms, std::bind(&cob_light::timer_callback, this));
  }

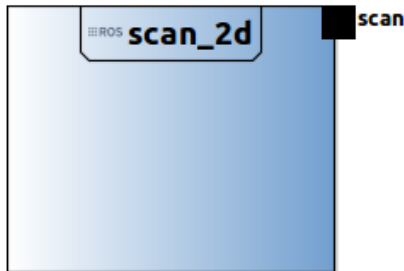
private:
  // Subscriber callback
  void command_callback(const cob_light::msg::ColorRGBAArray::SharedPtr msg) const {
    RCLCPP_INFO(this->get_logger(), "command topic got a message");
  }

  rclcpp::Subscription<cob_light::msg::ColorRGBAArray>::SharedPtr command_;
  // Subscriber callback
  void scan_unified_callback(const sensor_msgs::msg::LaserScan::SharedPtr msg) const {
    RCLCPP_INFO(this->get_logger(), "/scan_unified topic got a message");
  }
}
```

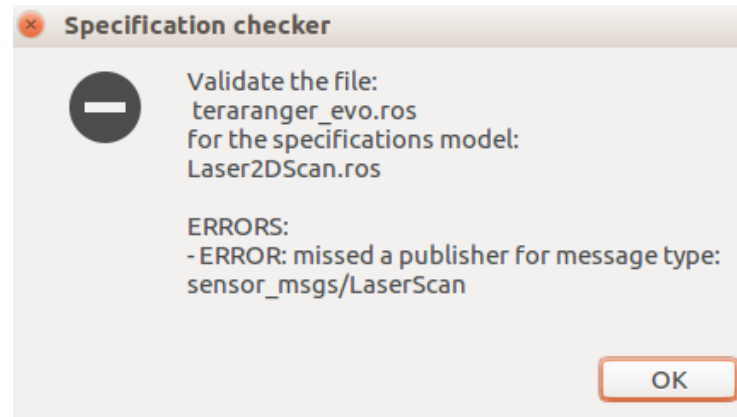

Benefits and applications

2) Identify common design patterns and check specification compliance

- Large scale analysis of components (web interface)
- Compare resulted models to extract common specification patterns
- Tooling includes a function to compare models (model to common specifications)



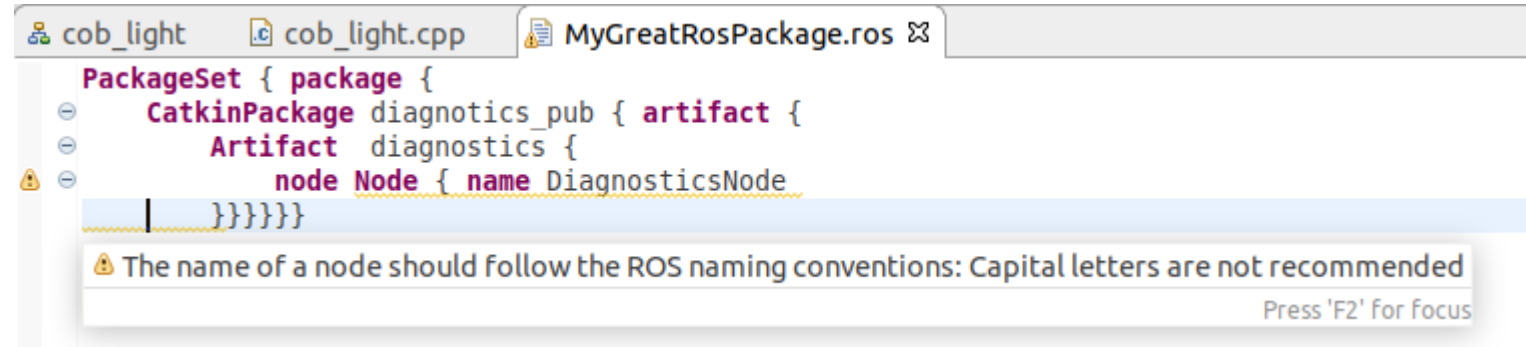
```
PackageSet { package {  
  CatkinPackage scan_2d { artifact {  
    Artifact scan_2d { node Node { name scan_2d  
      publisher {  
        Publisher { name scan message "sensor_msgs.LaserScan"  
      }  
    }  
  }  
}}
```



Benefits and applications

3) Diffuse best practices

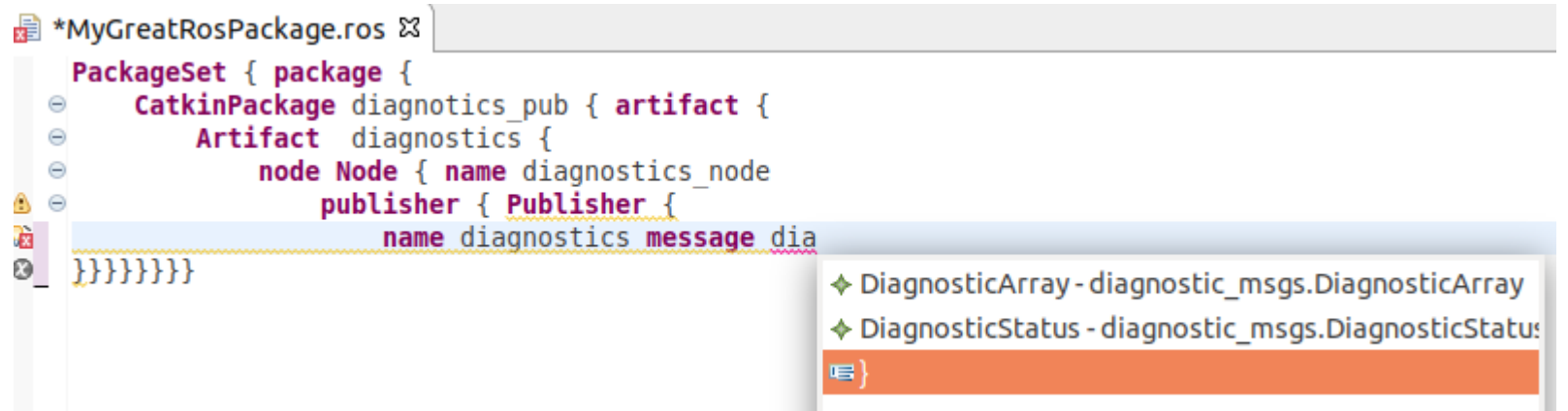
■ ROS naming conventions



The screenshot shows a code editor with two tabs: 'cob_light' and 'MyGreatRosPackage.ro'. The 'MyGreatRosPackage.ro' tab is active, displaying a ROS package configuration. The configuration includes a 'package' block with a 'diagnostics_pub' artifact, a 'diagnostics' artifact, and a 'Node' node. The 'Node' node has a 'name' field set to 'DiagnosticsNode'. A warning message is displayed below the code: 'The name of a node should follow the ROS naming conventions: Capital letters are not recommended'. The warning message also includes the instruction 'Press 'F2' for focus'.

```
PackageSet { package {
  CatkinPackage diagnostics_pub { artifact {
    Artifact diagnostics {
      node Node { name DiagnosticsNode
    }
  }
}
```

■ By default only load the common interfaces (msgs/srvs/actions)



The screenshot shows a code editor with a tab labeled '*MyGreatRosPackage.ro'. The 'MyGreatRosPackage.ro' tab is active, displaying a ROS package configuration. The configuration includes a 'package' block with a 'diagnostics_pub' artifact, a 'diagnostics' artifact, a 'diagnostics_node' node, and a 'diagnostics_message' publisher. The 'diagnostics_message' publisher has a 'name' field set to 'diagnostics_message'. A list of common interfaces is displayed below the code: 'DiagnosticArray - diagnostic_msgs.DiagnosticArray' and 'DiagnosticStatus - diagnostic_msgs.DiagnosticStatus'.

```
PackageSet { package {
  CatkinPackage diagnostics_pub { artifact {
    Artifact diagnostics {
      node Node { name diagnostics_node
      publisher { Publisher {
        name diagnostics_message dia
      }
    }
  }
}
```

Project Explorer

- de.fraunhofer.ipa.ros.communic
- scan_system_demo
 - Project Dependencies
 - components
 - rosnodes
 - diagnostic_aggregator.ros
 - scan_unifier.ros
 - sick_s300.ros
 - src-gen
 - scan_composition.launch
 - scan_compositioninstall.sh
 - representations.aird
 - scan_composition.rossystem

new RosSystem

scan_composition.launch

scan_front

scan_front/scan

diagnostics

aggregator_node

diagnostics

diagnostics_toplevel_state

scan_rear

scan_rear/scan

diagnostics

scan_unifier_node

scan1

scan2

scan_unified

Palette

- Tools
 - Topic Connection
 - Service Connection
 - ActionConnection
 - New Component

Properties

Console

Ros Publisher scan_rear/scan

Main

Semantic

Style

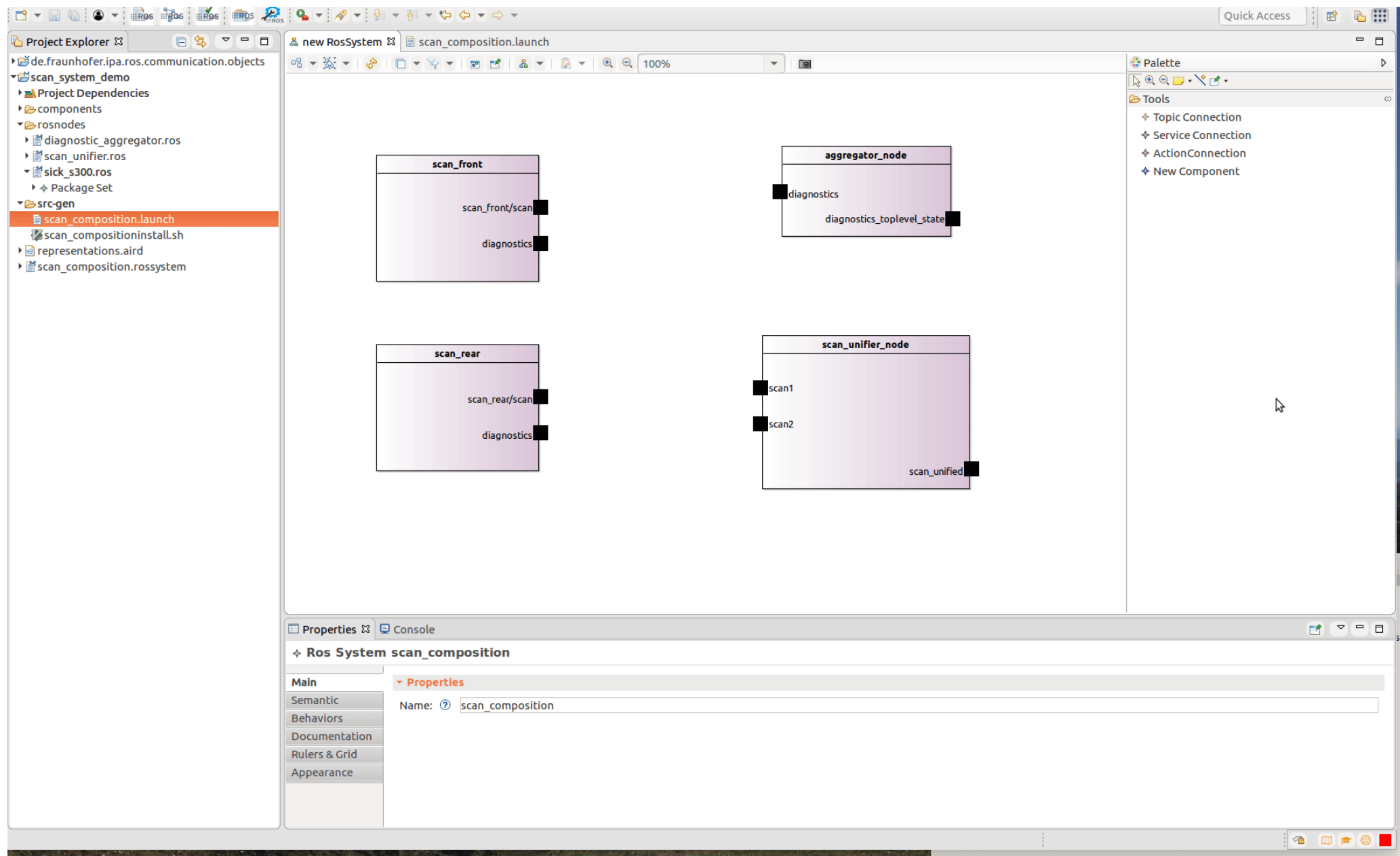
Appearance

Properties

Publisher: ? Publisher scan

Name: ? scan_rear/scan

Ns: ? scan_rear



Benefits and applications

5) Interoperability with other frameworks



Gefördert durch:



aufgrund eines Beschlusses
des Deutschen Bundestages

- Auto-translation of all the nodes or system to a generic concept of a component
 - Component defined as the interfaces that offer to interact with it (inputs and outputs) *
 - Allow modularity (systems of components/sub-systems/systems)

- Working example with SeRoNet
 - Model-to-model automatic transformation for communication objects (msgs and srvs) described using primitives: Int, Bool,String...
 - Model-to-model semi-automatic transformation for components ("bridge" from ROS interfaces))

Summary

- ✓ Reuse hand-written code - bottom-up approach
- ✓ Improve the understanding of what will happen at runtime
- ✓ Components composition and generate roslaunch files
- ✓ Check the use of common patterns/specifications
- ✓ Encapsulating ROS manually written "everyday code" in a formal structure for the interoperability with component-based frameworks

Future work

- Complete and complement HAROS
- ROS2 extension
 - Extractor support for ROS2 (HAROS latest release)
 - Generate ROS2 “launch” files
- Tooling eclipse-independent (e.g. web-service)

<https://github.com/ipa320/ros-model>

Bugs? Contributions? Ideas for new applications?



■ OPEN ISSUES: <https://github.com/ipa320/ros-model/issues>

■ CONTACT ME: M. Sc. Nadia Hammoudeh Garcia

Project Leader

Robot and Assistive Systems

nadia.hammoudeh.garcia@ipa.fraunhofer.de

<https://github.com/ipa-nhg>

THANKS!